

# Simplicial surfaces in GAP

Markus Baumeister

30.08.2017

- 1 General simplicial surfaces
- 2 Edge colouring and group properties
- 3 Abstract folding

- 1 General simplicial surfaces
- 2 Edge colouring and group properties
- 3 Abstract folding

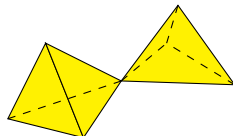
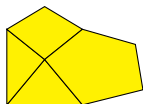
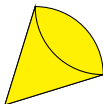
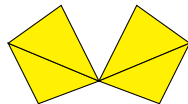
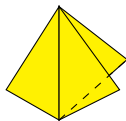
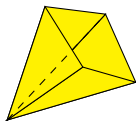
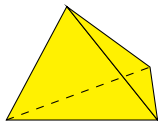
# Motivation

# Motivation

Goal: simplicial surfaces (and generalisations) in GAP

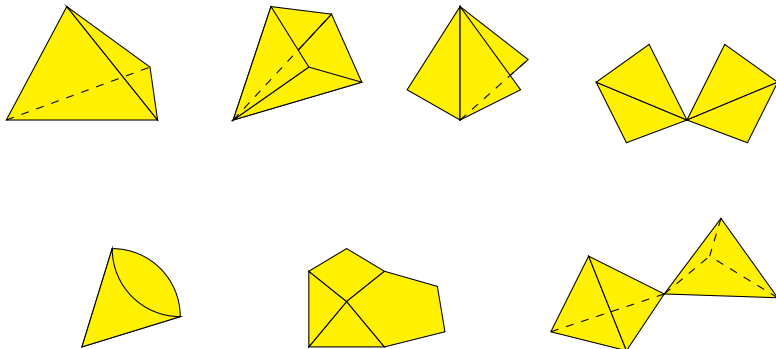
# Motivation

Goal: simplicial surfaces (and generalisations) in GAP



# Motivation

Goal: simplicial surfaces (and generalisations) in GAP



⇝ examples of **polygonal complexes**

# No embedding



# No embedding

We do **not** work with embeddings (mostly)

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute (compare the pentagon flippy)

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute (compare the pentagon flippy)
- if often unknown for an abstractly constructed surface

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute (compare the pentagon flip)
- is often unknown for an abstractly constructed surface
- is mostly independent from *intrinsic structure*

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute (compare the pentagon flippy)
  - if often unknown for an abstractly constructed surface
  - is mostly independent from *intrinsic structure*
- ⇒ lengths and angles are not important

# No embedding

We do **not** work with embeddings (mostly)

- is very hard to compute (compare the pentagon flippy)
- if often unknown for an abstractly constructed surface
- is mostly independent from *intrinsic structure*

⇒ lengths and angles are not important

↪ incidence structure is intrinsic

# Incidence structure of a polygonal complex



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$

•2 •3 •5 •7 •11

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$

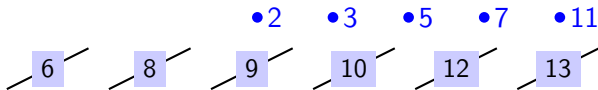
•2   •3   •5   •7   •11

- set of edges  $\mathcal{E}$

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

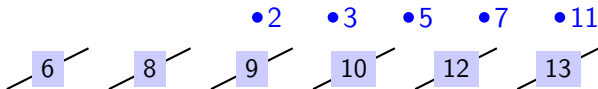
- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

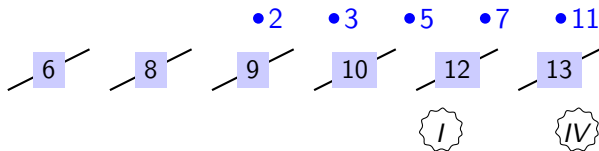
- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$



# Incidence structure of a polygonal complex

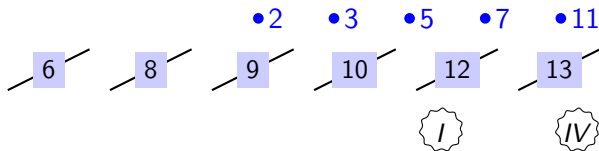
A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$

- set of edges  $\mathcal{E}$

- set of faces  $\mathcal{F}$

- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

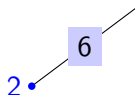
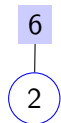
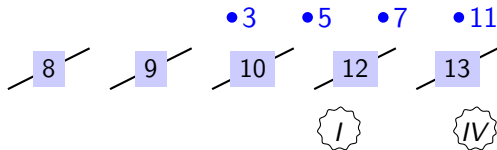




# Incidence structure of a polygonal complex

A **polygonal complex** consists of

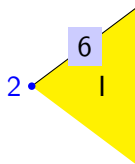
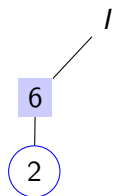
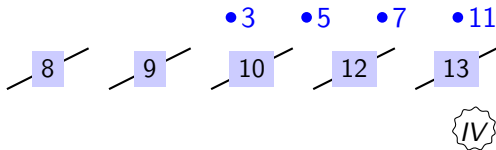
- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

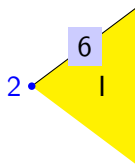
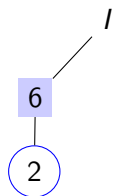
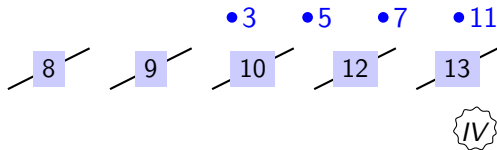
- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

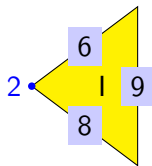
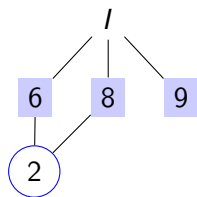
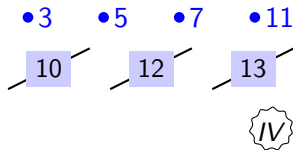


- 1 Every face is a polygon

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

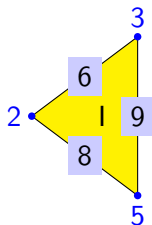
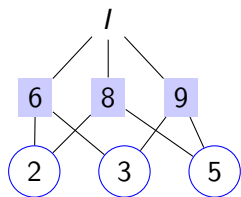
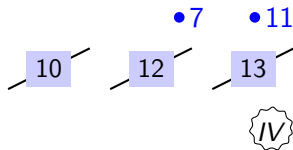


- 1 Every face is a polygon

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

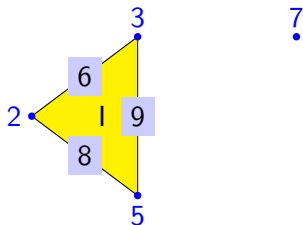
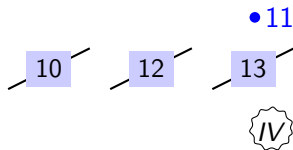
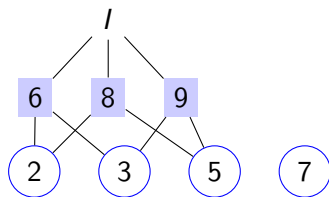


- 1 Every face is a polygon

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

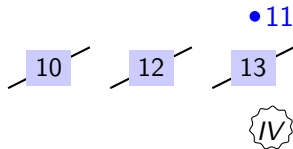
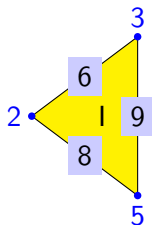
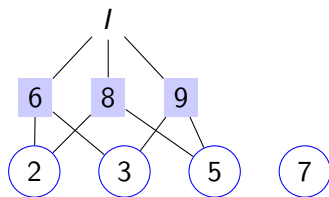


- 1 Every face is a polygon

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$



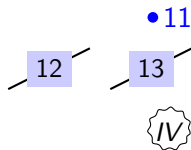
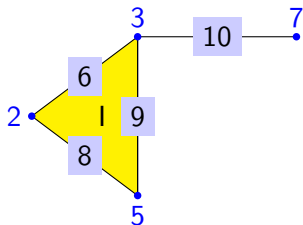
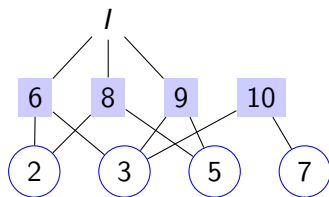
7 •

- 1 Every face is a polygon
- 2 Every vertex lies in an edge

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$



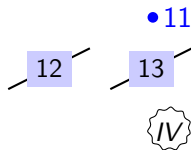
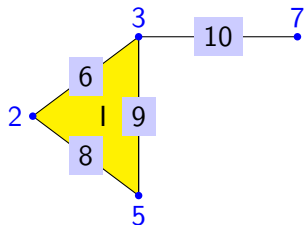
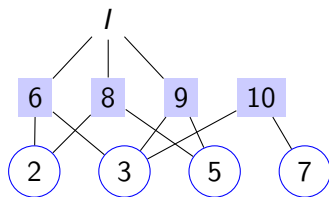
- 1 Every face is a polygon
- 2 Every vertex lies in an edge



# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

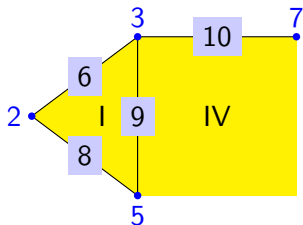
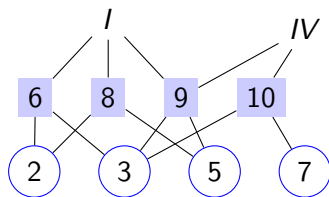
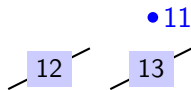


- 1 Every face is a polygon
- 2 Every vertex lies in an edge and every edge lies in a face

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$

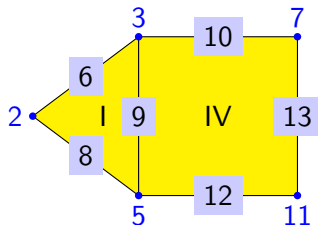
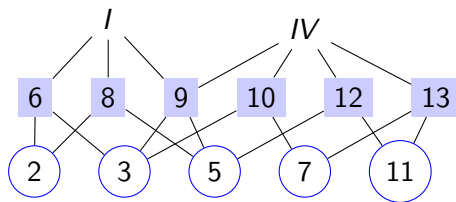


- 1 Every face is a polygon
- 2 Every vertex lies in an edge and every edge lies in a face

# Incidence structure of a polygonal complex

A **polygonal complex** consists of

- set of vertices  $\mathcal{V}$
- set of edges  $\mathcal{E}$
- set of faces  $\mathcal{F}$
- transitive relation  $\subseteq (\mathcal{V} \times \mathcal{E}) \uplus (\mathcal{V} \times \mathcal{F}) \uplus (\mathcal{E} \times \mathcal{F})$



- 1 Every face is a polygon
- 2 Every vertex lies in an edge and every edge lies in a face

# Isomorphism testing

# Isomorphism testing

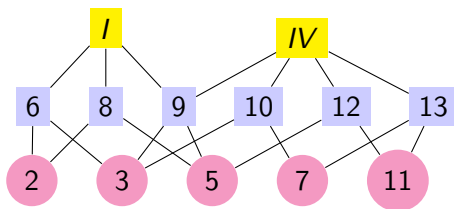
Incidence geometry allows “easy” isomorphism testing.

# Isomorphism testing

Incidence geometry allows “easy” isomorphism testing. Incidence structure can be interpreted as a coloured graph:

# Isomorphism testing

Incidence geometry allows “easy” isomorphism testing. Incidence structure can be interpreted as a coloured graph:



# Isomorphism testing

Incidence geometry allows “easy” isomorphism testing. Incidence structure can be interpreted as a coloured graph:

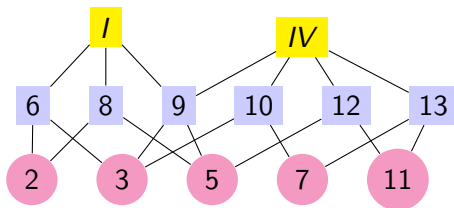


$\rightsquigarrow$  reduce to graph isomorphism problem



# Isomorphism testing

Incidence geometry allows “easy” isomorphism testing. Incidence structure can be interpreted as a coloured graph:



↪ reduce to graph isomorphism problem

Solved by NautyTracesInterface (by Gutsche, Niemeyer, Schweitzer)

# General properties

# General properties

Some properties can be computed for all polygonal complexes:

# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity

# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity
- Euler–Characteristic

# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity
- Euler–Characteristic

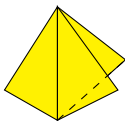
*Orientability* is **not** one of them.

# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity
- Euler–Characteristic

*Orientability* is **not** one of them. Counterexample:



# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity
- Euler–Characteristic

*Orientability* is **not** one of them. Counterexample:



⇒ every edge lies in at most two faces (for well–definedness)



# General properties

Some properties can be computed for all polygonal complexes:

- Connectivity
- Euler–Characteristic

*Orientability* is **not** one of them. Counterexample:



⇒ every edge lies in at most two faces (for well-definedness)

⇔ **ramified polygonal surfaces**

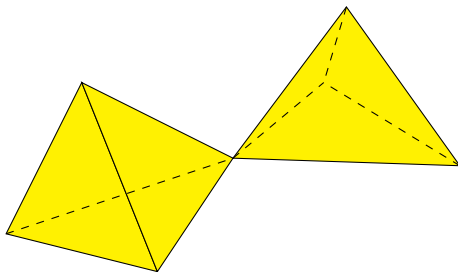
# Why ramified?

# Why ramified?

Typical example of ramified polygonal surface:

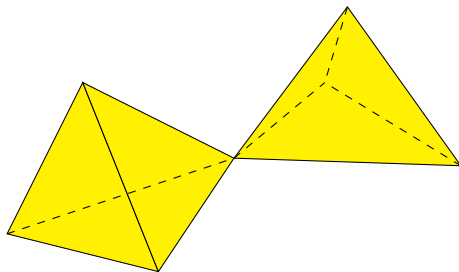
# Why ramified?

Typical example of ramified polygonal surface:



# Why ramified?

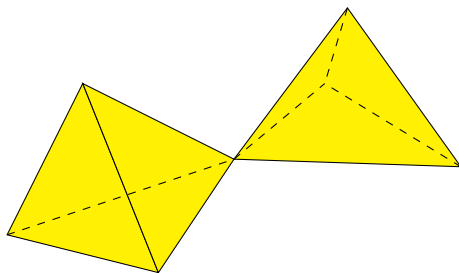
Typical example of ramified polygonal surface:



⇒ It is not a surface – there is a *ramification* at the central vertex

# Why ramified?

Typical example of ramified polygonal surface:



⇒ It is not a surface – there is a *ramification* at the central vertex  
A **polygonal surface** does not have these ramifications.

- 1 General simplicial surfaces
- 2 Edge colouring and group properties
- 3 Abstract folding

# Embedding question



# Embedding question

Given: A polygonal complex

# Embedding question

Given: A polygonal complex

- Can it be embedded?

# Embedding question

Given: A polygonal complex

- Can it be embedded?
- In how many ways?

# Embedding question

Given: A polygonal complex

- Can it be embedded?
- In how many ways?

Simplifications:

# Embedding question

Given: A polygonal complex

- Can it be embedded?
- In how many ways?

Simplifications:

- 1 Only polygonal surfaces (surface that is build from polygons)

# Embedding question

Given: A polygonal complex

- Can it be embedded?
- In how many ways?

Simplifications:

- 1 Only polygonal surfaces (surface that is build from polygons)
- 2 All polygons are triangles (**simplicial surfaces**)

# Embedding question

Given: A polygonal complex

- Can it be embedded?
- In how many ways?

Simplifications:

- 1 Only polygonal surfaces (surface that is build from polygons)
- 2 All polygons are triangles (**simplicial surfaces**)
- 3 All triangles are isometric

# Embedding question

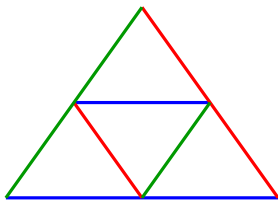
Given: A polygonal complex

- Can it be embedded?
- In how many ways?

Simplifications:

- 1 Only polygonal surfaces (surface that is build from polygons)
- 2 All polygons are triangles (**simplicial surfaces**)
- 3 All triangles are isometric

↪ Edge-colouring encodes different lengths





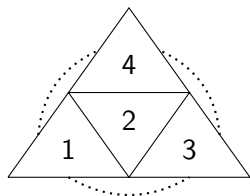
# Colouring as permutation

# Colouring as permutation

Consider tetrahedron

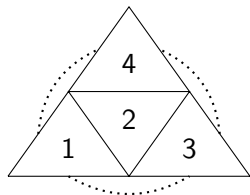
# Colouring as permutation

Consider tetrahedron



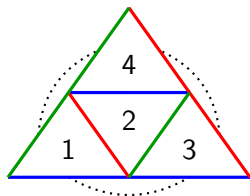
# Colouring as permutation

Consider tetrahedron with edge colouring



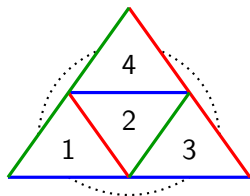
# Colouring as permutation

Consider tetrahedron with edge colouring



# Colouring as permutation

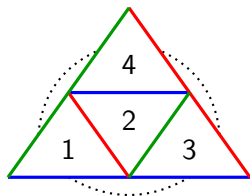
Consider tetrahedron with edge colouring



*simplicial surface*  $\Rightarrow$

# Colouring as permutation

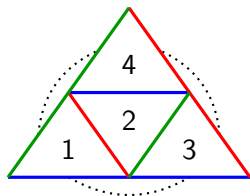
Consider tetrahedron with edge colouring



*simplicial surface*  $\Rightarrow$  at most two faces at each edge

# Colouring as permutation

Consider tetrahedron with edge colouring



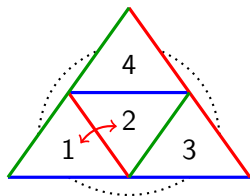
*simplicial surface*  $\Rightarrow$  at most two faces at each edge

$\rightsquigarrow$  every edge defines transposition of incident faces



# Colouring as permutation

Consider tetrahedron with edge colouring



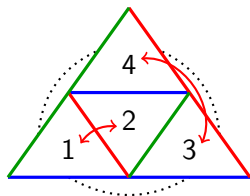
*simplicial surface*  $\Rightarrow$  at most two faces at each edge

$\rightsquigarrow$  every edge defines transposition of incident faces

- (1,2)

# Colouring as permutation

Consider tetrahedron with edge colouring



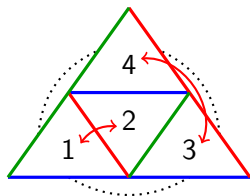
*simplicial surface*  $\Rightarrow$  at most two faces at each edge

$\rightsquigarrow$  every edge defines transposition of incident faces

- $(1,2)(3,4)$

# Colouring as permutation

Consider tetrahedron with edge colouring

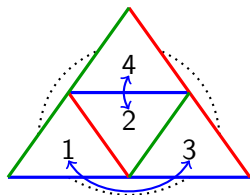


*simplicial surface*  $\Rightarrow$  at most two faces at each edge

- $\rightsquigarrow$  every edge defines transposition of incident faces
- $\rightsquigarrow$  every colour class defines permutation of the faces
  - $(1,2)(3,4)$

# Colouring as permutation

Consider tetrahedron with edge colouring

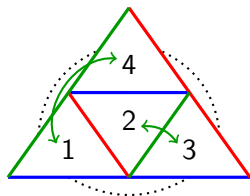


*simplicial surface*  $\Rightarrow$  at most two faces at each edge

- $\rightsquigarrow$  every edge defines transposition of incident faces
- $\rightsquigarrow$  every colour class defines permutation of the faces
  - $(1,2)(3,4)$  ,  $(1,3)(2,4)$

# Colouring as permutation

Consider tetrahedron with edge colouring

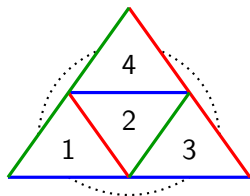


*simplicial surface*  $\Rightarrow$  at most two faces at each edge

- $\rightsquigarrow$  every edge defines transposition of incident faces
- $\rightsquigarrow$  every colour class defines permutation of the faces
  - $(1,2)(3,4)$  ,  $(1,3)(2,4)$  ,  $(1,4)(2,3)$

# Colouring as permutation

Consider tetrahedron with edge colouring

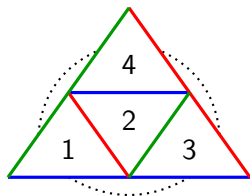


*simplicial surface*  $\Rightarrow$  at most two faces at each edge

- $\rightsquigarrow$  every edge defines transposition of incident faces
- $\rightsquigarrow$  every colour class defines permutation of the faces
  - $(1,2)(3,4)$  ,  $(1,3)(2,4)$  ,  $(1,4)(2,3)$
- $\rightsquigarrow$  group theoretic considerations

# Colouring as permutation

Consider tetrahedron with edge colouring

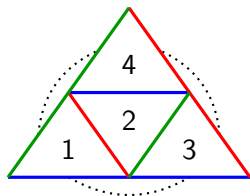


*simplicial surface*  $\Rightarrow$  at most two faces at each edge

- $\rightsquigarrow$  every edge defines transposition of incident faces
- $\rightsquigarrow$  every colour class defines permutation of the faces
  - $(1,2)(3,4)$  ,  $(1,3)(2,4)$  ,  $(1,4)(2,3)$
- $\rightsquigarrow$  group theoretic considerations
  - The connected components of the surface correspond to

# Colouring as permutation

Consider tetrahedron with edge colouring



*simplicial surface*  $\Rightarrow$  at most two faces at each edge

$\rightsquigarrow$  every edge defines transposition of incident faces

$\rightsquigarrow$  every colour class defines permutation of the faces

•  $(1,2)(3,4)$  ,  $(1,3)(2,4)$  ,  $(1,4)(2,3)$

$\rightsquigarrow$  group theoretic considerations

- ▶ The connected components of the surface correspond to the orbits of  $\langle \sigma_a, \sigma_b, \sigma_c \rangle$  on the faces



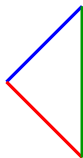
# How do faces fit together?

# How do faces fit together?

Consider a face of the surface

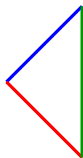
# How do faces fit together?

Consider a face of the surface



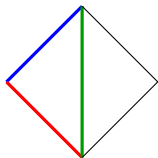
# How do faces fit together?

Consider a face of the surface and a neighbouring face



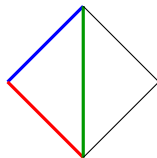
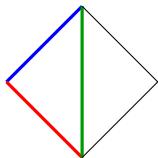
# How do faces fit together?

Consider a face of the surface and a neighbouring face



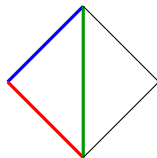
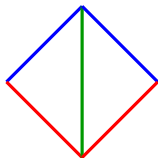
# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



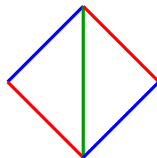
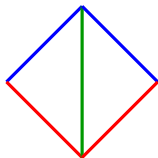
# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



# How do faces fit together?

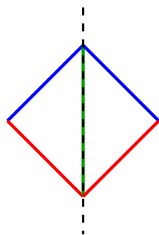
Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



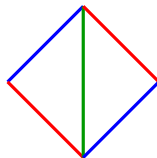


# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:

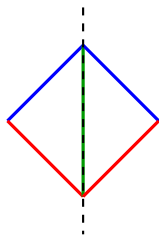


mirror (m)

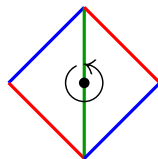


# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



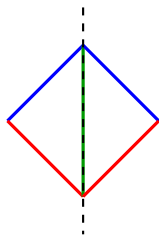
mirror (m)



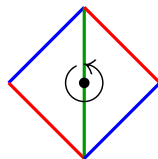
rotation (r)

# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



mirror (m)

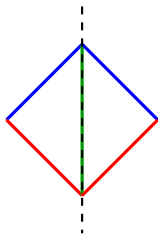


rotation (r)

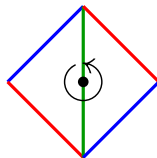
This gives an **mr-assignment** for the edges.

# How do faces fit together?

Consider a face of the surface and a neighbouring face  
The neighbour can be coloured in two ways:



mirror (m)



rotation (r)

This gives an **mr-assignment** for the edges.  
Permutations and mr-assignment uniquely determine the surface.

# Constructing surfaces from groups

# Constructing surfaces from groups

A general mr-assignment leads to complicated surfaces.

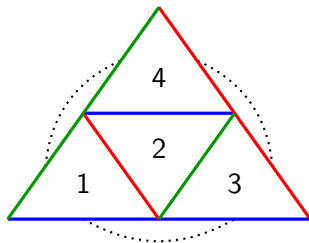
# Constructing surfaces from groups

A general mr-assignment leads to complicated surfaces.  
Simplification: edges of same colour have the same type

# Constructing surfaces from groups

A general mr-assignment leads to complicated surfaces.  
Simplification: edges of same colour have the same type

Example

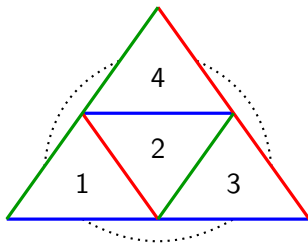




# Constructing surfaces from groups

A general mr-assignment leads to complicated surfaces.  
Simplification: edges of same colour have the same type

Example

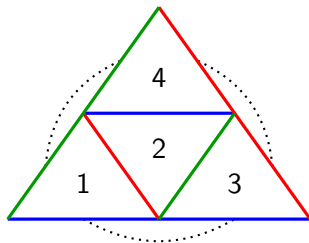


has an rrr-structure

# Constructing surfaces from groups

A general mr-assignment leads to complicated surfaces.  
Simplification: edges of same colour have the same type

Example



has an rrr-structure

The easiest structure is an mmm-structure.

# Covering

# Covering

We want to characterize surfaces where all edges are mirrors.

# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle*

# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle, i. e. there is an incidence-preserving map to the simplicial surface consisting of exactly one face.*

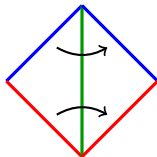
# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle, i. e. there is an incidence-preserving map to the simplicial surface consisting of exactly one face.*

Consider



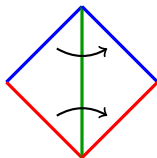
# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle, i. e. there is an incidence-preserving map to the simplicial surface consisting of exactly one face.*

Consider



⇒ Unique map that preserves incidence



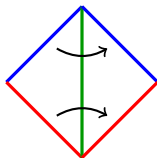
# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle, i. e. there is an incidence-preserving map to the simplicial surface consisting of exactly one face.*

Consider



⇒ Unique map that preserves incidence

- Covering pulls back a mirror-colouring of the triangle.

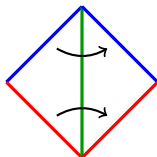
# Covering

We want to characterize surfaces where all edges are mirrors.

## Lemma

*A simplicial surface has an mmm-structure iff it covers a single triangle, i. e. there is an incidence-preserving map to the simplicial surface consisting of exactly one face.*

Consider



⇒ Unique map that preserves incidence

- Covering pulls back a mirror-colouring of the triangle.
- Mirror-colouring defines a map to the triangle.

# Construction from permutations

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions*

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions



# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions

# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions
- The vertices are

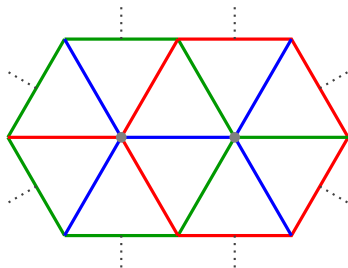
# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions
- The vertices are



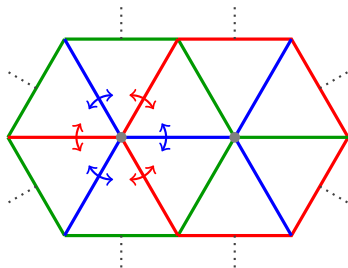
# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions
- The vertices are



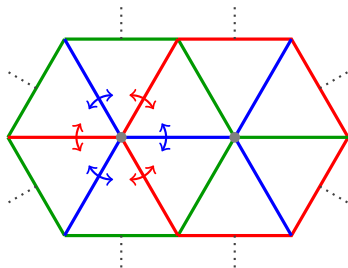
# Construction from permutations

Start with three involutions  $\sigma_a$ ,  $\sigma_b$ ,  $\sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions
- The vertices are the orbits of  $\langle \sigma_a, \sigma_b \rangle$  on the faces



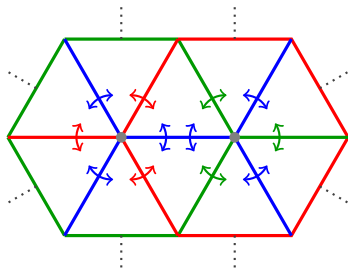
## Construction from permutations

Start with three involutions  $\sigma_a, \sigma_b, \sigma_c$  (like generators of a finite group)

## Lemma

*There exists a coloured surface with the given involutions where all edges are mirror edges.*

- The faces are the points moved by the involutions
- The edges are the cycles of the involutions
- The vertices are the orbits of  $\langle \sigma_a, \sigma_b \rangle$  on the faces (for all pairs)



# Construction example

# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$



# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

## Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

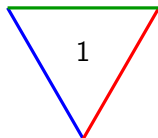
$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

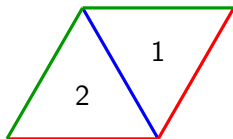


## Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

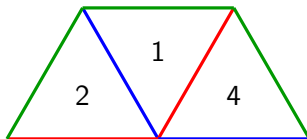


## Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

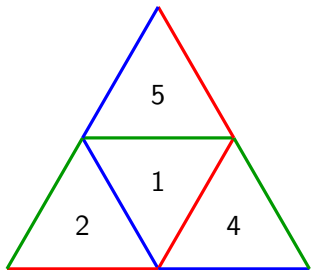


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

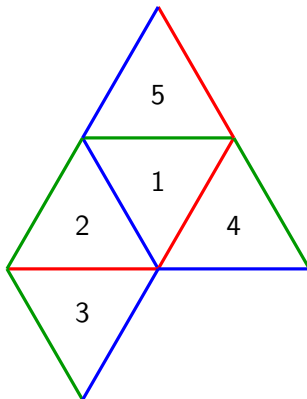


## Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

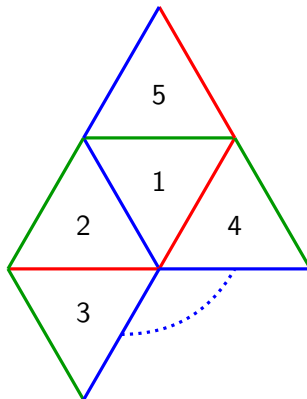


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$



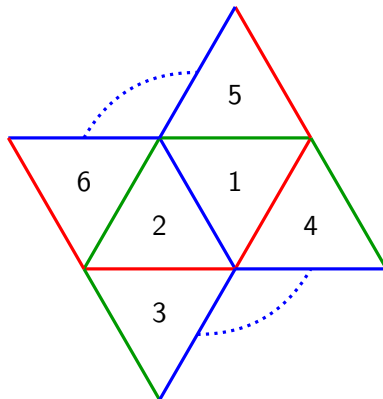


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

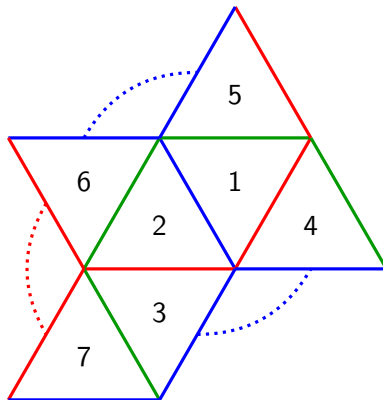


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

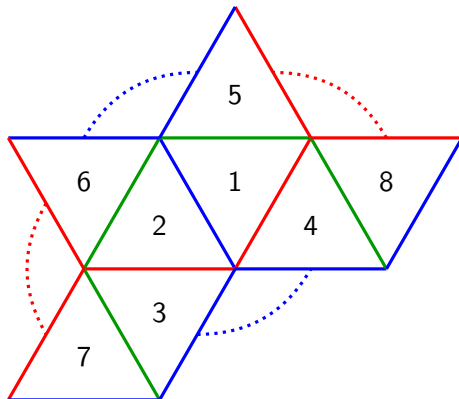


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

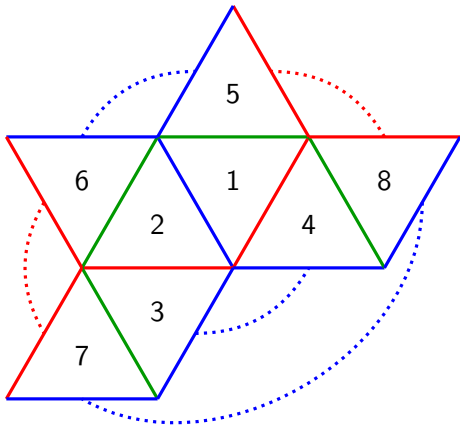


## Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$

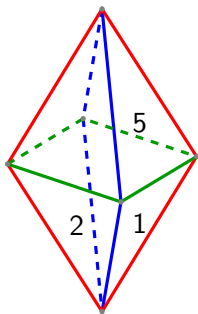
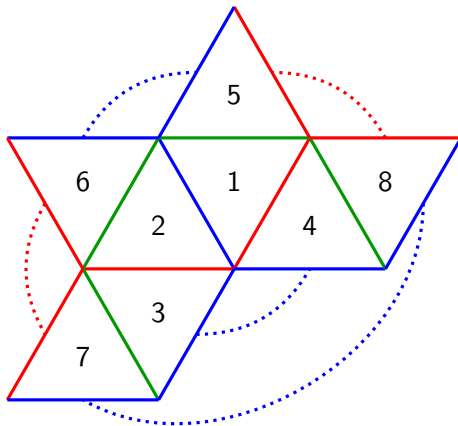


# Construction example

$$\sigma_a = (1, 2)(3, 4)(5, 6)(7, 8)$$

$$\sigma_b = (1, 4)(2, 3)(5, 8)(6, 7)$$

$$\sigma_c = (1, 5)(2, 6)(3, 7)(4, 8)$$



- 1 General simplicial surfaces
- 2 Edge colouring and group properties
- 3 Abstract folding

# What kind of folding?

# What kind of folding?

There are many different kinds of folding (e. g. Origami)



# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

- Folding of surface in  $\mathbb{R}^3$

# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

- Folding of surface in  $\mathbb{R}^3$
- Possible folding edges are fixed

# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

- Folding of surface in  $\mathbb{R}^3$
- Possible folding edges are fixed
- Folding should be rigid (no curvature)

# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

- Folding of surface in  $\mathbb{R}^3$
- Possible folding edges are fixed
- Folding should be rigid (no curvature)

Goal: Classify possible folding patterns (given a net)

# What kind of folding?

There are many different kinds of folding (e. g. Origami)

Here:

- Folding of surface in  $\mathbb{R}^3$
- Possible folding edges are fixed
- Folding should be rigid (no curvature)

Goal: Classify possible folding patterns (given a net)

# Why are embeddings hard?

# Why are embeddings hard?

Ideally, we would like to have embeddings.



## Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

## Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)

# Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)
- We can only show foldability for specific small examples

# Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)
- We can only show foldability for specific small examples
  - ▶ Usually using regularity (like crystallographic symmetry)

# Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)
- We can only show foldability for specific small examples
  - ▶ Usually using regularity (like crystallographic symmetry)
  - ▶ No general method

# Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)
- We can only show foldability for specific small examples
  - ▶ Usually using regularity (like crystallographic symmetry)
  - ▶ No general method
- It is very hard to define iterated folding in an embedding

# Why are embeddings hard?

Ideally, we would like to have embeddings.

But we want to define folding independently from an embedding, since:

- They are very hard to compute (even for small examples)
- We can only show foldability for specific small examples
  - ▶ Usually using regularity (like crystallographic symmetry)
  - ▶ No general method
- It is very hard to define iterated folding in an embedding

# Folding without embedding



# Folding without embedding

Central idea:

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)
  - ▶ allows abstraction from embedding

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)
  - ▶ allows abstraction from embedding

↪ Incidence geometry (polygonal complex/surface)

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)
  - ▶ allows abstraction from embedding

⇒ Incidence geometry (polygonal complex/surface)

- Captures some folding restrictions



# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)
  - ▶ allows abstraction from embedding

↪ Incidence geometry (polygonal complex/surface)

- Captures some folding restrictions (rigidity of tetrahedron)

# Folding without embedding

Central idea:

- Don't model folding process (needs embedding)
- Describe starting and final folding state
  - ▶ Only consider changes in the topology (like identification of faces)
  - ▶ allows abstraction from embedding

⇒ Incidence geometry (polygonal complex/surface)

- Captures some folding restrictions (rigidity of tetrahedron)
- Still needs a lot of refinement

# Important properties of folding

# Important properties of folding

- The class of surfaces is not closed under folding

# Important properties of folding

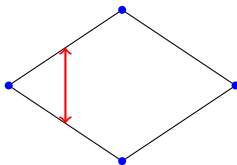
- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*

# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces

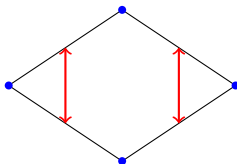
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces



# Important properties of folding

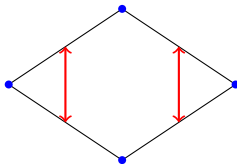
- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces





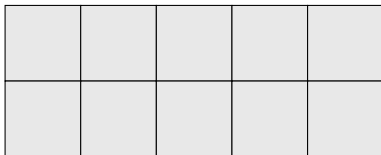
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces



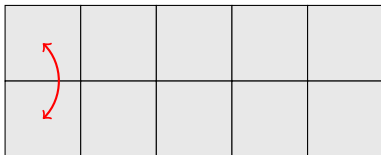
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces



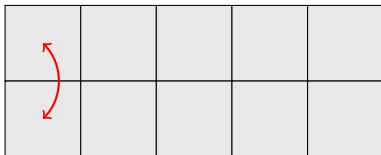
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces



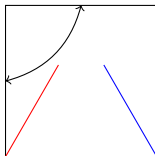
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces
  - ▶ The forced identification is not unique



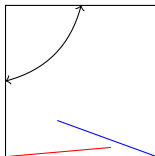
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces
  - ▶ The forced identification is not unique



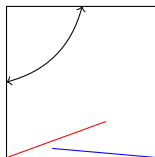
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces
  - ▶ The forced identification is not unique



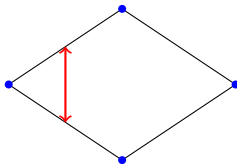
# Important properties of folding

- The class of surfaces is not closed under folding
- Folding can be undone by *unfolding*
- Identification of two faces might force identification of two other faces
  - ▶ Can apply to arbitrary many faces
  - ▶ The forced identification is not unique



# Important properties of folding

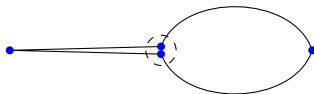
- The class of surfaces is not closed under folding
  - Folding can be undone by *unfolding*
  - Identification of two faces might force identification of two other faces
    - ▶ Can apply to arbitrary many faces
    - ▶ The forced identification is not unique
- ⇒ Identify only two faces at a time





# Important properties of folding

- The class of surfaces is not closed under folding
  - Folding can be undone by *unfolding*
  - Identification of two faces might force identification of two other faces
    - ▶ Can apply to arbitrary many faces
    - ▶ The forced identification is not unique
- ⇒ Identify only two faces at a time



# How to define abstract folding?

# How to define abstract folding?

We need to define two structures:

# How to define abstract folding?

We need to define two structures:

- 1 A folding state

# How to define abstract folding?

We need to define two structures:

- 1 A folding state
- 2 The folding steps

# How to define abstract folding?

We need to define two structures:

- ① A folding state
  - ▶ Based on polygonal complexes
  
- ② The folding steps

# How to define abstract folding?

We need to define two structures:

- ① A folding state
  - ▶ Based on polygonal complexes
  - ▶ Describe “is folded together” by an equivalence relation
- ② The folding steps

# How to define abstract folding?

We need to define two structures:

- ① A folding state
  - ▶ Based on polygonal complexes
  - ▶ Describe “is folded together” by an equivalence relation
  - ▶ Describe order of faces in folding state
- ② The folding steps



# How to define abstract folding?

We need to define two structures:

- ① A folding state
  - ▶ Based on polygonal complexes
  - ▶ Describe “is folded together” by an equivalence relation
  - ▶ Describe order of faces in folding state
- ② The folding steps
  - ▶ Only two faces at a time

# How to define abstract folding?

We need to define two structures:

- ① A folding state
  - ▶ Based on polygonal complexes
  - ▶ Describe “is folded together” by an equivalence relation
  - ▶ Describe order of faces in folding state
- ② The folding steps
  - ▶ Only two faces at a time
  - ▶ Explain “unordered folding” (e. g. covering)

# How to define abstract folding?

We need to define two structures:

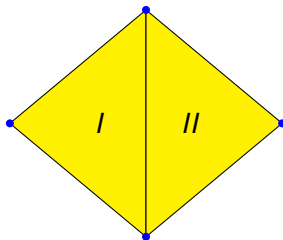
- ① A folding state
  - ▶ Based on polygonal complexes
  - ▶ Describe “is folded together” by an equivalence relation
  - ▶ Describe order of faces in folding state
- ② The folding steps
  - ▶ Only two faces at a time
  - ▶ Explain “unordered folding” (e. g. covering)
  - ▶ Modify to include face order relations

# Unordered Folding (Covering)

# Unordered Folding (Covering)

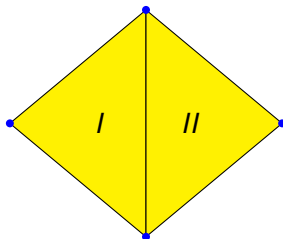
Why do we need more than a polygonal complex?

# Unordered Folding (Covering)



Why do we need more than a polygonal complex?

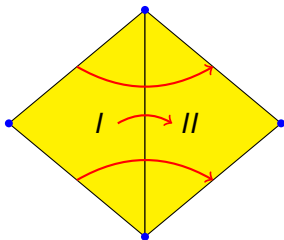
# Unordered Folding (Covering)



Why do we need more than a polygonal complex?

Naive folding definition: surjective map that respects incidence

# Unordered Folding (Covering)

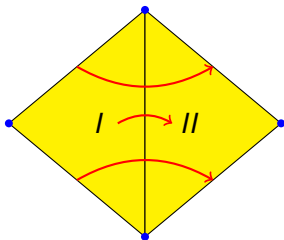


Why do we need more than a polygonal complex?

Naive folding definition: surjective map that respects incidence



# Unordered Folding (Covering)

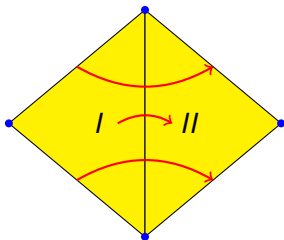


Why do we need more than a polygonal complex?

Naive folding definition: surjective map that respects incidence

Problem: Can't be unfolded

# Unordered Folding (Covering)



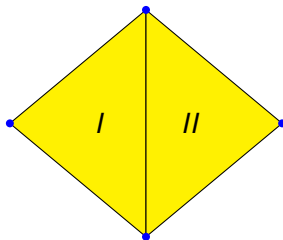
Why do we need more than a polygonal complex?

Naive folding definition: surjective map that respects incidence

Problem: Can't be unfolded

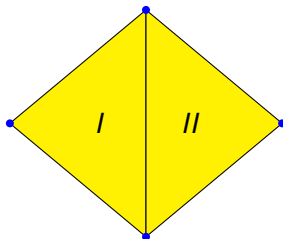
⇒ Folding state should not forget original structure

# Unordered Folding (Covering)



Represent folding by equivalence relation

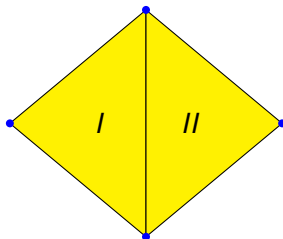
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces

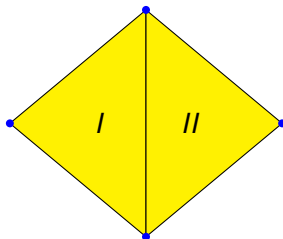
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together

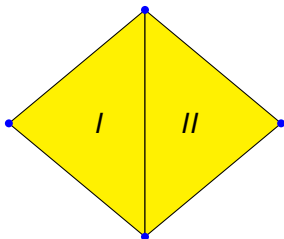
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent,

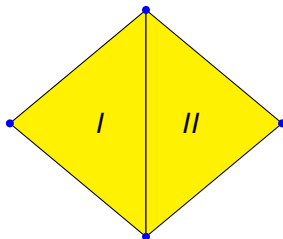
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent, then their vertices have to be as well

# Unordered Folding (Covering)

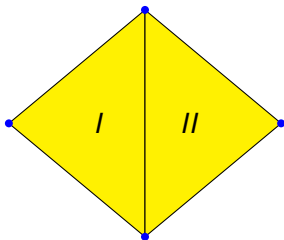


Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent, then their vertices have to be as well (likewise for faces)



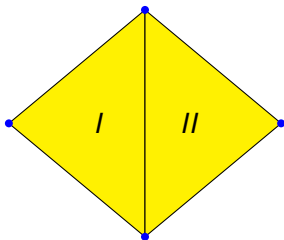
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent, then their vertices have to be as well (likewise for faces)
- The vertices of an edge are not equivalent

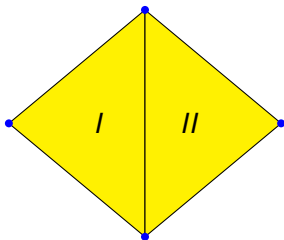
# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent, then their vertices have to be as well (likewise for faces)
- The vertices of an edge are not equivalent (likewise for faces)

# Unordered Folding (Covering)



Represent folding by equivalence relation

- Separate relation on vertices, edges and faces
- Two elements are equivalent if they are folded together
- If two edges are equivalent, then their vertices have to be as well (likewise for faces)
- The vertices of an edge are not equivalent (likewise for faces)

⇒ Unordered folding is coarsening of equivalence relation

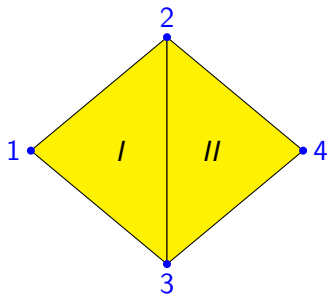
# How does folding work?

# How does folding work?

- 1 Choose two faces that are not folded together

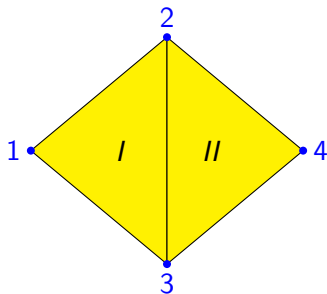
# How does folding work?

- 1 Choose two faces that are not folded together



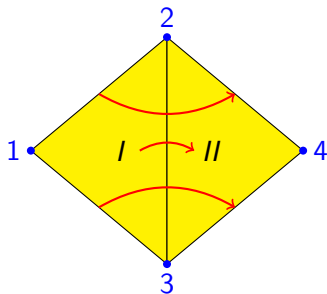
# How does folding work?

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them



# How does folding work?

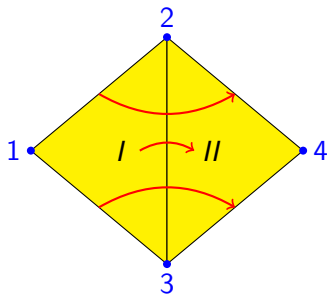
- 1 Choose two faces that are not folded together
- 2 Choose how to identify them





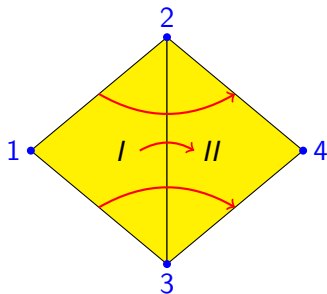
# How does folding work?

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them (like  $I \sim II$  and  $1 \sim 4$ )



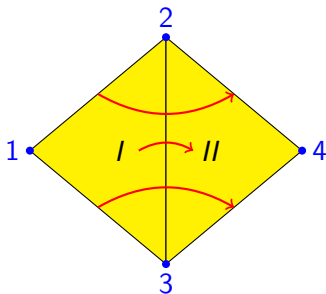
# How does folding work?

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them (like  $I \sim II$  and  $1 \sim 4$ )
- 3 Add those pairs to the equivalence relation



# How does folding work?

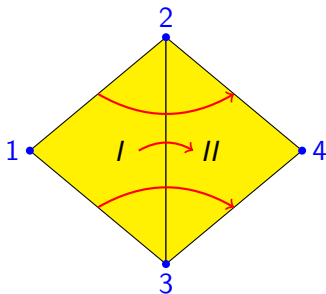
- 1 Choose two faces that are not folded together
- 2 Choose how to identify them (like  $I \sim II$  and  $1 \sim 4$ )
- 3 Add those pairs to the equivalence relation



Only restriction:

# How does folding work?

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them (like  $I \sim II$  and  $1 \sim 4$ )
- 3 Add those pairs to the equivalence relation

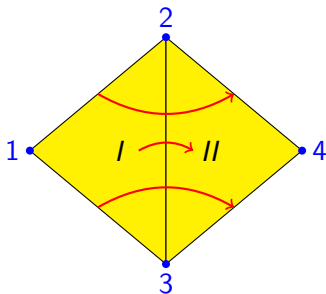


Only restriction:

Two vertices in an edge can't be identified

# How does folding work?

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them (like  $I \sim II$  and  $1 \sim 4$ )
- 3 Add those pairs to the equivalence relation



Only restriction:

Two vertices in an edge can't be identified (slightly generalized)

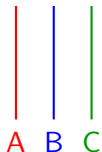
# Limitation of unordered folding

# Limitation of unordered folding

We can't work with ordering of faces:

# Limitation of unordered folding

We can't work with ordering of faces:





# Limitation of unordered folding

We can't work with ordering of faces:



Adding a linear order on each face equivalence class

# Limitation of unordered folding

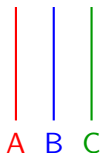
We can't work with ordering of faces:



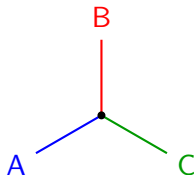
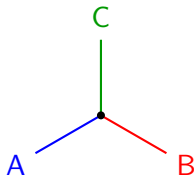
Adding a linear order on each face equivalence class is not enough:

# Limitation of unordered folding

We can't work with ordering of faces:

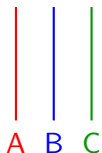


Adding a linear order on each face equivalence class is not enough:

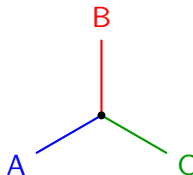
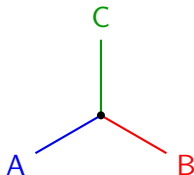


# Limitation of unordered folding

We can't work with ordering of faces:



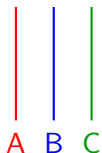
Adding a linear order on each face equivalence class is not enough:



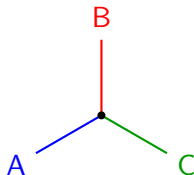
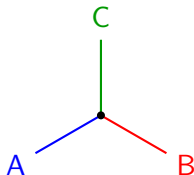
$\leadsto$  define order of faces around edges

# Limitation of unordered folding

We can't work with ordering of faces:



Adding a linear order on each face equivalence class is not enough:



$\leadsto$  define order of faces around edges (we will skip the details)

# Folding complex

# Folding complex

## Definition

A **folding complex**

# Folding complex

## Definition

A **folding complex** is a *polygonal complex* together with



# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- 1 An equivalence relation on vertices, edges and faces

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① An equivalence relation on vertices, edges and faces (“is folded together”)

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class*

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible*

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

- linear orderings get concatenated



# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

- linear orderings get concatenated
- cyclical orderings are opened at one point

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

- linear orderings get concatenated
- cyclical orderings are opened at one point and combined

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

- linear orderings get concatenated
- cyclical orderings are opened at one point and combined
- !! compatibility is not easily transferred,

# Folding complex

## Definition

A **folding complex** is a polygonal complex together with

- ① *An equivalence relation on vertices, edges and faces (“is folded together”)*
- ② *A linear ordering on each face equivalence class*
- ③ *A cyclical ordering of the faces around each edge equivalence class such that the orderings are compatible (in an appropriate sense).*

To identify faces with each other, we have to combine those orderings.

- linear orderings get concatenated
- cyclical orderings are opened at one point and combined
- !! compatibility is not easily transferred, but can be calculated

# Changed definition of folding

# Changed definition of folding

Before (no ordering):

# Changed definition of folding

Before (no ordering):

- 1 Choose two faces that are not folded together

## Changed definition of folding

Before (no ordering):

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation



# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation

# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation

$\leadsto$  Each face has two sides

# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation

$\leadsto$  Each face has two sides

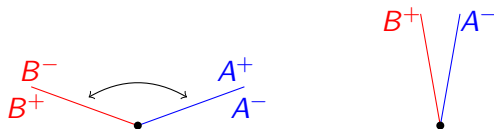


# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation

$\leadsto$  Each face has two sides

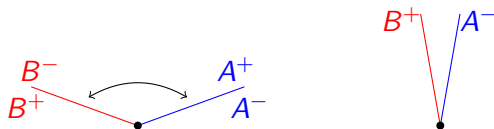


# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation
- 3 Choose the sides of the faces that will meet

↪ Each face has two sides

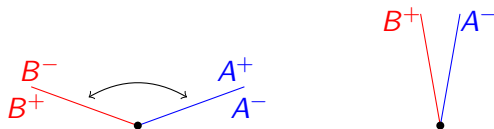


# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation
- 3 Choose the sides of the faces that will meet and modify the orderings

↪ Each face has two sides

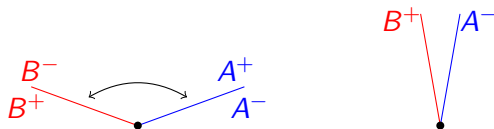


# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation
- 3 Choose the sides of the faces that will meet and modify the orderings

↪ Each face has two sides



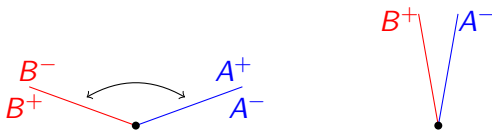
⇒ Define folding by two face sides (**folding plan**)

# Changed definition of folding

Folding with ordering:

- 1 Choose two faces that are not folded together
- 2 Choose how to identify them and extend the equivalence relation
- 3 Choose the sides of the faces that will meet and modify the orderings

↪ Each face has two sides



⇒ Define folding by two face sides (**folding plan**)

↪ Allows reversible (un)folding



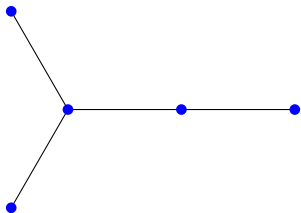
# Structure of multiple foldings

# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes

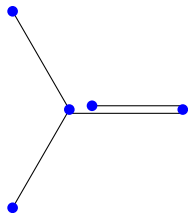
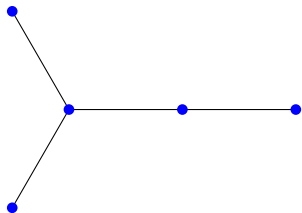
# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes



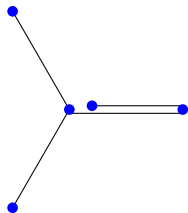
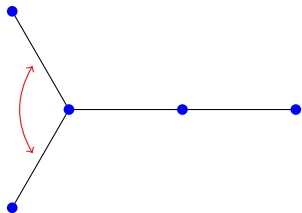
# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes



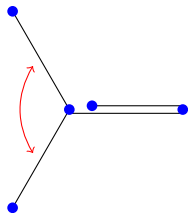
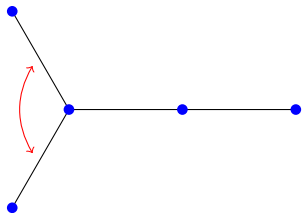
# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes



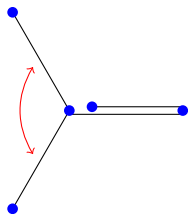
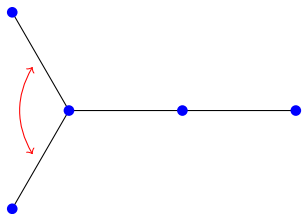
# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes



# Structure of multiple foldings

With folding plans we can perform the same folding in different folding complexes



$\rightsquigarrow$  more structure on the set of possible foldings

# Folding graph



# Folding graph

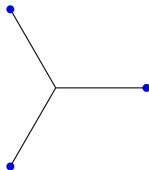
- Vertices are folding complexes (modelling folding states)

# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes

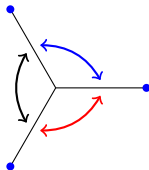
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



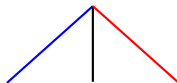
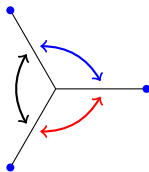
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



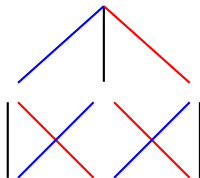
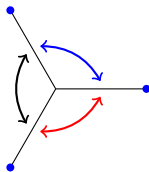
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



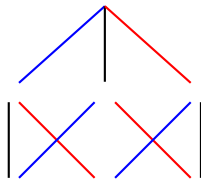
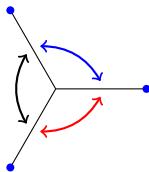
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



# Folding graph

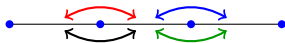
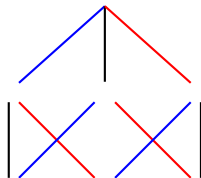
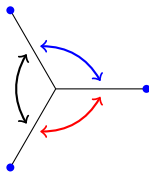
- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes





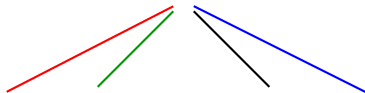
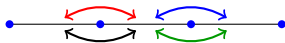
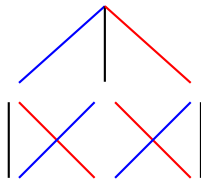
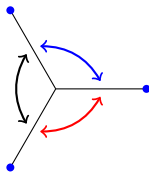
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



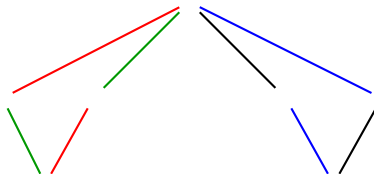
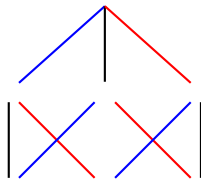
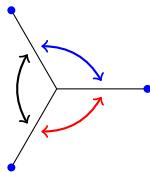
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



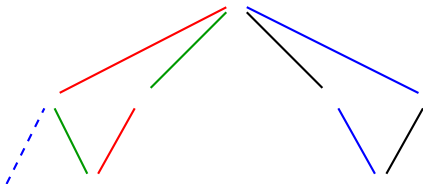
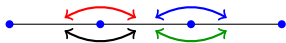
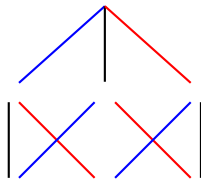
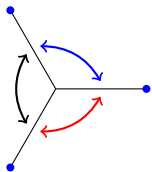
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



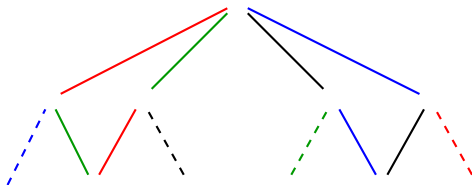
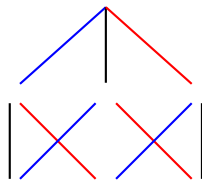
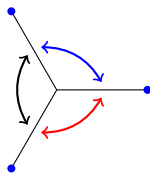
# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



# Folding graph

- Vertices are folding complexes (modelling folding states)
- Edges are folding plans connecting two folding complexes



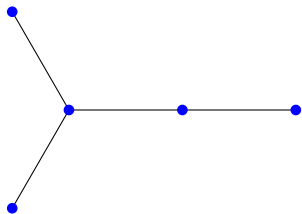
# Drawback of folding plans

# Drawback of folding plans

Some foldings that “should” be the same, aren’t:

# Drawback of folding plans

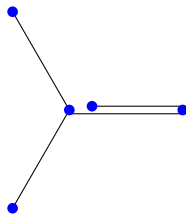
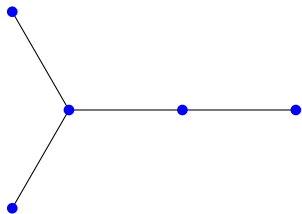
Some foldings that “should” be the same, aren't:





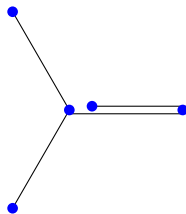
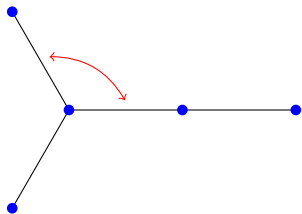
# Drawback of folding plans

Some foldings that “should” be the same, aren’t:



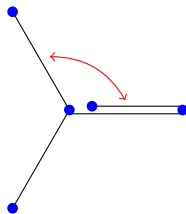
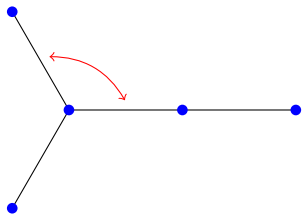
# Drawback of folding plans

Some foldings that “should” be the same, aren't:



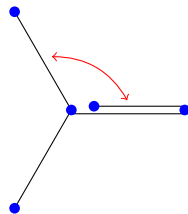
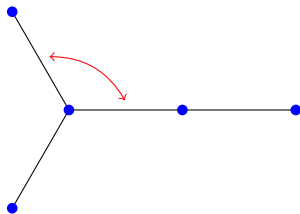
# Drawback of folding plans

Some foldings that “should” be the same, aren't:



# Drawback of folding plans

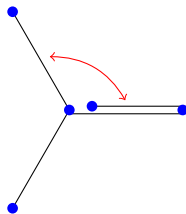
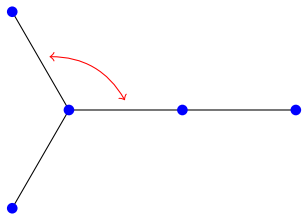
Some foldings that “should” be the same, aren’t:



⇒ If you know the folding structure of a small complex,

# Drawback of folding plans

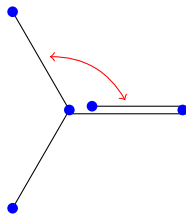
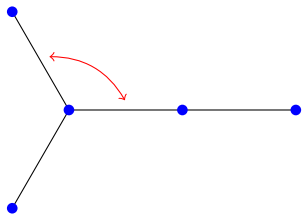
Some foldings that “should” be the same, aren’t:



⇒ If you know the folding structure of a small complex, you can’t easily find the folding structure of an extended complex

# Drawback of folding plans

Some foldings that “should” be the same, aren't:



- ⇒ If you know the folding structure of a small complex, you can't easily find the folding structure of an extended complex
- ⇝ Folding plans are not optimal to model folding.

# Questions?